**9.集成学习**

**9.1概念**

集成学习（ensemble learning）通过构建并组合多个学习器来完成学习任务。集成学习通过将多个学习器进行结合，常获得比单一学习器显著优越的泛化性能。根据个体学习器是否是同类型的学习器（由同一个算法生成，比如C4.5,BP等），分为同质和异质。同质的个体学习器又叫做基学习器，而异质的个体学习器则直接成为个体学习器。

**（1）集成学习的原则**

要获得比单一学习器更好的性能。个体学习器应该好而不同，即个体学习器应该具有一定的准确性，不能差于弱学习器，并且具有多样性，即学习器之间有差异。不同的模型通常会在测试集上产生不同的误差；如果成员的误差是独立的，集成模型将显著地比其成员表现更好。

**（2）集成学习的类别**

（a）个体学习器之间存在强依赖关系、必须串行生成的序列化方法。代表是Boosting。

（b）个体学习器之间不存在强依赖关系、可同时生成的并行化方法。代表是Bagging。

集成学习基本属于以下三种范式：

①Bagging。Bagging基于并行策略：基学习器之间不存在依赖关系，可同时生成。由于训练数据不同，我们的基学习器可望具有较大的差异。Bagging主要关注降低方差。

Bagging的基本工作机制：

1）利用自助采样法对训练集随机采样，重复进行 T 次；

自助采样法：对 m 个样本的训练集，有放回的采样 m 次；此时，样本在 m 次采样中始终没被采样的概率约为 0.368，即每次自助采样只能采样到全部样本的 63.2% 左右。

2）基于每个采样集训练一个基学习器，并得到 T 个基学习器；

3）预测时，集体投票决策。Bagging在预测输出进行结合时，对于分类问题，采用简单投票法；对于回归问题，采用简单平均法。

Bagging的优点：1）高效。Bagging集成与直接训练基学习器的复杂度同阶；2）Bagging能不经修改的适用于多分类、回归任务；3）包外估计。使用剩下的样本作为验证集进行包外估计（out-of-bag estimate）。

Bagging特点：1）训练每个基学习器时只使用一部分样本；2）偏好不稳定的学习器作为基学习器（所谓不稳定的学习器，指的是对样本分布较为敏感的学习器）。

②Boosting。基于串行策略，基学习器之间存在依赖关系，新的学习器需要根据上一个学习器生成。Boosting意为提升，意思是希望将每个弱学习器提升为强学习器。Boosting关注的主要是降低偏差。

Boosting的基本工作机制：

1）先从初始训练集中学习一个基学习器，初始训练集中各样本的权重是相同的；

2）根据上一个基学习器的表现，调整样本权重，使分类错误的样本得到更多的关注（提高错误样本的权重）；

3）基于调整后的样本分布，训练下一个基学习器；

4）如此反复，直到基学习器数目达到T，最终将这T个基学习器进行加权结合。

Boosting的特点：每次学习都会使用全部训练样本；代表算法：AdaBoost 算法、GBDT 算法。

③Stacking：基于串行策略：初级学习器与次级学习器之间存在依赖关系，初学习器的输出作为次级学习器的输入。

Stacking的基本工作机制：

1）先从初始训练集训练 T 个不同的初级学习器;

2）利用每个初级学习器的输出构建一个次级数据集，该数据集依然使用初始数据集的标签；

3）根据新的数据集训练次级学习器；

4）多级学习器的构建过程类似。

5）为了降低过拟合的风险，一般会利用交叉验证的方法使不同的初级学习器在不完全相同的子集上训练。

以 k-折交叉验证为例：

- 初始训练集 D={(xi, yi)} 被划分成 D1, D2, .., Dk；

- 记 ht 表示第 t 个学习器，并在除 Dj 外的数据上训练；

- 当ht 训练完毕后，有 zi,t = ht (xi)；

- T 个初级学习器在xi 上共产生 T 个输出；

- 这 T 个输出共同构成第 i 个次级训练数据 zi = (zi,1, zi,2, ..., zi,T)，标签依然为yi；

- 在 T 个初级学习器都训练完毕后，得到次级训练集 D'={( zi, yi)}

**（3）集成学习基学习器的选择原则**

集成学习的基学习器需要选择不稳定的学习器，比如决策树，这是因为：

（a）不稳定的学习器容易受到样本分布的影响（方差大），很好的引入了随机性；这有助于在集成学习（特别是采用 Bagging 策略）中提升模型的泛化能力。

（b）为了更好的引入随机性，有时会随机选择一个属性子集中的最优分裂属性，而不是全局最优（随机森林）。

**（4）Boosting/Bagging与偏差/方差的关系。**

简单来说，Boosting 能提升弱分类器性能的原因是降低了偏差；Bagging 则是降低了方差。Boosting 的基本思路就是在不断减小模型的训练误差（拟合残差或者加大错类的权重），加强模型的学习能力，从而减小偏差；但 Boosting 不会显著降低方差，因为其训练过程中各基学习器是强相关的，缺少独立性。Bagging 方法对 n 个独立不相关的模型预测结果取平均，方差是原来的 1/n；假设所有基分类器出错的概率是独立的，超过半数基分类器出错的概率会随着基分类器的数量增加而下降。

**9.2 AdaBoost算法**

**9.2.1 AdaBoost算法解决Boosting两个基本问题的方法**

（1）每一轮如何改变数据的权值或概率分布。——开始时，每个样本的权值是一样的，AdaBoost的做法是提高上一轮弱分类器错误分类样本的权值，同时降低那些被正确分类样本的权值。

（2）如何将弱分类器组合成一个强分类器。—— AdaBoost 采取加权表决的方法（加法模型）。具体的，AdaBoost 会加大分类误差率小的基学习器的权值，使其在表决中起到更大的作用，同时减小分类误差率大的基学习器的权值。

**9.2.2AdaBoost算法流程**

输入：训练集 T={(x1, y1)，(x2, y2)，…，(xN, yN)}, xi ∈ Rn，yi ∈ {-1,+1}；弱学习算法。

输出：最终学习器G(*x*)

（1）初始化训练数据的权值分布

（2）对m=1, 2,…, M

（a）使用权值分布为Dm的训练集学习，得到基分类器

（b）计算Gm(*x*)在训练集上的分类误差率(实际上分类误差率就等于所有分类错误的数据的权值之和)

*I*(*x*) 为指示函数，若G(*x*) != *y*为真，则*I*(G(*x*) != *y*)=1，反之为0。

（c）计算 Gm(*x*) 的系数：

（d）更新训练集的权值分布

其中Zm是规范化因子，使 Dm+1 成为一个概率分布，类似 Softmax 函数。

因为 *y*, G(*x*) ∈ {-1, 1}，所以实际上：

因此 wm+1,i 也可以写作

（3）构建基学习器的线性组合

得到最终的分类器

AdaBoost算法说明：

1）开始时，假设训练集中所有数据具有均匀的权值分布，这一假设保证第一步能够在原始数据上学习基本分类器G1(x)；

2）计算分类误差率，实际上就是计算所有分类错误的数据的权值之和。

3）Gm(*x*) 的系数 αm 表示该学习器在最终学习器中的重要性；公式表明当分类错误率em ≤ 时，αm ≥ 0，并且αm随em的减小而增大，即分类误差率越小的基本分类器在最终分类器中的作用越大。

4）被基分类器分类错误的样本权值会扩大，而分类正确的权值会缩小——不改变训练数据，而不断改变训练数据权值的分布，使训练数据在基学习器的学习中起到不同的作用。

5）线性组合f(x)实现M个基本分类器的加权表决。所有系数αm之和不为1，f(x)的符合决定实例x的类，f(x)的绝对值表示分类的确信度。

AdaBoost由三部分组成：1）分类器权重更新公式；2）样本分布（也就是样本权重）更新公式；3）加性模型。 最小化指数损失函数。

AdaBoost优点：1）不改变所给的训练数据，而不断改变训练数据的权值分布，使得训练数据在基本分类器的学习中起不同的作用；2）利用基本分类器的加权线性组合构建最终分类器；3）AdaBoost被实践证明是一种很好的防止过拟合的方法（但至今为什么至今没从理论上证明）。

**9.2.3 AdaBoost算法的训练误差分析**

AdaBoost最基本的性质是它能在学习过程中不断减少训练误差，即在训练数据集上的分类误差率。

（1）AdaBoost的训练误差界

AdaBoost算法最终分类器的训练误差界为

以上定理说明，可以在每一轮选取适当的Gm使得Zm最小，从而使训练误差下降最快。

（2）二分类问题AdaBoost的训练误差界

推论：如果存在γ > 0，对所有m有γm ≥ γ，则

表明在此条件下，AdaBoost的训练误差是以**指数速率**下降的。

注意，AdaBoost算法不需要知道下界γ。与早期的一些提升方法不同，AdaBoost具有适应性，即它能适应弱分类器各自的训练误差率。这也是它的名称（适应的提升）的由来。

**9.2.4 AdaBoost算法的解释**

AdaBoost算法还有另一个解释，即可以认为AdaBoost算法是模型为加法模型、损失函数为指数函数、学习算法为前向分步算法的二分类学习方法。

定义加法模型（additive model）：

其中b(x; γ)为基函数，γ为基函数的参数；β为基函数的系数。

在给定训练数据和损失函数L(y, f(x))的情况下，学习加法模型相当于损失函数的最小化问题：

通常上式的优化是一个复杂的优化问题。前向分步算法求解这一加法模型优化问题的想法是：如果能够从前向后，每一步只学习一个基函数及其系数，逐步逼近优化目标函数。

前向分步算法流程：

输入：训练集 T={(x1, y1),..,(xN, yN)}，损失函数 L(y, f(x))，基函数集 {b(x; γ)}

输出：加法模型 *f*(*x*)。

1）初始化 *f*0(*x*)=0

2）对 m=1,2,..,M

（a）极小化损失函数，得到 (βm, γm)

（b）更新模型 *f*m(*x*)

3）得到加法模型

前向分步算法将同时求解m=1,2,..,M所有参数(βm, γm)的问题简化为逐次求解各(βm, γm)的优化问题——思想上有点像梯度下降。

AdaBoost算法是前向分步加法算法的特例，这时模型是由基本分类器组成的加法模型，损失函数是指数函数L(y, *f*(*x*)) = exp(-y*f*(*x*))。

**9.3提升树算法**

提升树是以决策树（分类树或回归树）为基本分类器的提升方法，通常是二叉分类树或二叉回归树，被认为是统计学习中性能最好的方法之一。提升树模型可以表示为决策树的加法模型：

其中，T(x；Θm)表示决策树，Θm为决策树的参数，M为树的个数。

提升树采用前向分步算法，首先确定初始提升树f0(x) = 0，第m步的模型是

fm(x) = fm-1(x) + T(x；Θm)

其中fm-1(x)为当前模型，并通过经验风险极小化确定下一棵决策树的参数Θm：

由于树的线性组合可以很好的拟合训练数据，即使数据中的输入和输出之间的关系很复杂也是如此，所以提升树是一个高功能的学习算法。

根据使用的损失函数的不同，可以分成不同的提升树学习算法：1）回归问题算法，使用平方差损失函数；2）分类问题算法，使用指数损失函数；3）一般决策问题算法，使用一般损失函数。

（1）分类问题算法。分类问题只需将AdaBoost算法的基本分类器限制为二分类树即可，这时的提升树算法是AdaBoost算法的特殊情况。

（2）回归问题算法。回归问题损失函数采用平方差损失

这里r = y – fm-1(x)是当前模型拟合数据的残差（residual）。所以回归问题的提升树算法只需简单地拟合当前模型的残差。

回归问题的提升树算法：

输入：训练集 T={(x1,y1),..,(xN, yN)}, xi ∈ Rn, yi∈R；

输出：回归提升树 *f*M(*x*)。

1）初始化*f*0(*x*)=0；

2）对 m=1,2,..,M

i. 计算残差

ii. 拟合残差学习下一个回归树的参数

得到T(x；Θm)

iii. 更新 *f*m(*x*)

3）得到回归提升树

（3）一般决策问题。当损失函数是平方差损失或指数损失函数时，每一步优化都很简单，但是一般的损失函数优化并不容易。Freidman提出梯度提升（gradient boosting）算法，这是利用最速下降法的近似方法，其关键是利用损失函数的负梯度在当前模型的值作为回归问题提升树算法中的残差的近似值拟合一个回归树

**9.3.1梯度提升算法（GBDT）：**

输入：训练数据集 T={(x1,y1),..,(xN, yN)}, xi ∈ Rn, yi∈R；损失函数L(y，f(x))；

输出：回归树(x)。

1）初始化

2）对m = 1，2，…，M

（a）对i = 1，2，…，N，计算

（b）对rm,i拟合一个回归树，得第m棵树叶结点区域Rmj，j = 1，2，…，J

（c）对j = 1，2，…，J，计算

（d）更新

3）得到回归树

**1）算法说明：**

①算法第1步初始化，估计使损失函数最小的常数值，得到一棵只有一个根节点的树。

②第2(a)步计算损失函数的负梯度，将其作为残差的估计；对平方损失而言，负梯度就是残差；对于一般的损失函数，它是残差的近似。

③第2(b)步估计回归树的节点区域，以拟合残差的近似值。

④第2(c)步利用线性搜索估计叶节点区域的值，使损失函数极小化。

**2）梯度提升和梯度下降的区别和联系**

（a）联系：两者都是在每一轮迭代中，利用损失函数相对于模型的负梯度方向的信息来对当前模型进行更新。

（b）区别：在梯度下降中，模型是以参数化形式表示，从而模型的更新等价于参数的更新。而在梯度提升中，模型并不需要进行参数化表示，而是直接定义在函数空间中，从而大大扩展了可以使用的模型种类。

**9.3.2 GBDT的解释**

1）GB：梯度提升 Gradient Boosting

首先Boosting是一种集成方法。通过对弱分类器的组合得到强分类器，他是串行的，几个弱分类器之间是依次训练的。GBDT的核心就在于，**每一颗树学习的是之前所有树结论和的残差**。Gradient体现在：无论前面一颗树的cost function是什么，是均方差还是均差，只要它以误差作为衡量标准，那么残差向量都是它的全局最优方向，这就是Gradient。

2）DT：回归树 Regression Decision Tree

GDBT中的树**全部都是回归树**，核心就是**累加所有树的结果作为最终结果**。只有回归树的结果累加起来才是有意义的，分类的结果加是没有意义的。GDBT调整之后可以用于分类问题，但是内部还是回归树。

3）Shrinkage（Shrinkage（缩减）是GBDT算法的一个重要演进分支，目前大部分的源码都是基于这个版本的。）

（a）Shrinkage核心思想：Shrinkage认为每次走一小步来逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易防止过拟合。也就是说，它不信任每次学习到的残差，它认为每棵树只学习到了真理的一小部分，累加的时候只累加一小部分，通过多学习几棵树来弥补不足。

（b）Shrinkage具体的做法：仍然以残差作为学习目标，但是对于残差学习出来的结果，只累加一小部分（step\*残差）逐步逼近目标，step一般都比较小0.01-0.001,导致各个树的残差是渐变而不是陡变的。

本质上，Shrinkage为每一颗树设置了一个weight，累加时要乘以这个weight，但和Gradient没有关系。这个weight就是step。跟AdaBoost一样，Shrinkage能减少过拟合也是经验证明的，目前还没有理论证明。

4）GBDT适用范围。

（a）GBDT可以适用于回归问题（线性和非线性），相对于logistic regression仅能用于线性回归，GBDT适用面更广；

（b）GBDT也可用于二分类问题（设定阈值，大于为正，否则为负）和多分类问题。

5）GBDT的优点和局限性。

（a）优点：①预测阶段的计算速度快，树与树之间可并行化计算；②在分布稠密的数据集上，泛化能力和表达能力都很好；③采用决策树作为弱分类器使得GBDT模型具有较好的解释性和鲁棒性，能够自动发现特征间的高阶关系。

（b）缺点：①GBDT在高维稀疏的数据集上，表现不如支持向量机或者神经网络；②GBDT在处理文本分类特征问题上，相对其他模型的优势不如它在处理数值特征时明显；③训练过程需要串行训练，只能在决策树内部采用一些局部并行的手段提高训练速度。